

DEVOPS-DAY-4

Kubernetes (K8s)

Kubernetes is an open source container orchestration engine for automating deployment, scaling, and management of containerized applications. The open source project is hosted by the Cloud Native Computing Foundation (CNCF).

It provides a scalable and resilient framework for automating the deployment, scaling, and management of applications across clusters of servers.

A SMALL HISTORY OF K8S:

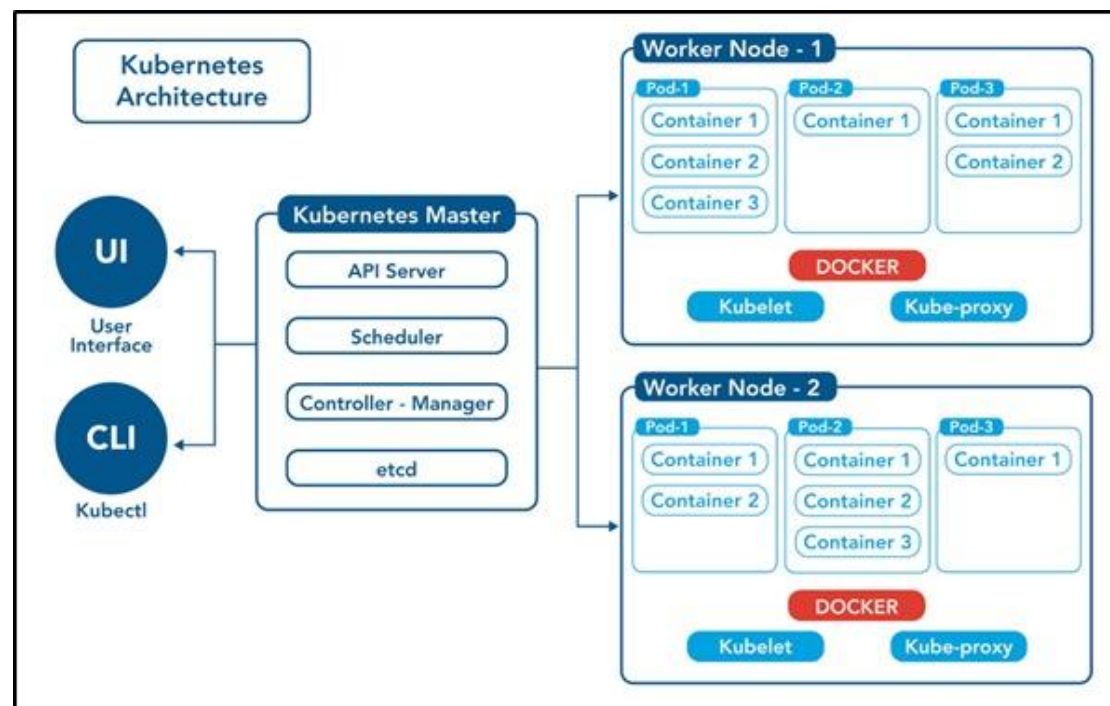
In the early 2000s, Google started developing a system called Borg to manage their internal containerized applications.

Borg enabled Google to run applications at scale, providing features such as automatic scaling, service discovery, and fault tolerance.

In 2014, Google open-sourced a version of Borg called Kubernetes.

Kubernetes was donated to the Cloud Native Computing Foundation (CNCF), a neutral home for open-source cloud-native projects, in July 2015.

Kubernetes 1.8 added significant enhancements for storage, security, and networking. Key features included the stable release of the stateful sets API, expanded support for volume plugins, and improvements in security policies.



Control Plane /Master Node

The control plane's components make global decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events (for example, starting up a new pod when a deployment's replicas field is unsatisfied).

Control plane components can be run on any machine in the cluster. Do not run user containers on this machine.

Node Components / Worker Nodes

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

1. **Master Node:** The master node is responsible for managing the cluster and coordinating the overall state of the system. It includes the following components:
 - a. **API Server:** The API server is the central control point for all interactions with the cluster. It exposes the Kubernetes API and handles requests from users and other components.
 - b. **Scheduler:** The scheduler is responsible for assigning workloads (pods) to individual worker nodes based on resource requirements, constraints, and other policies.
 - c. **Controller Manager:** The controller manager runs various controllers that monitor the cluster state and drive it towards the desired state. Examples include the replication controller, node controller, and service controller.
 - d. **etcd:** etcd is a distributed key-value store used by Kubernetes to store cluster state and configuration data.

Pod: The basic building block of Kubernetes. A pod represents a single instance of a running process within the cluster. It can encapsulate one or more containers that share the same network and storage resources.

Create a pod using run command

```
$ kubectl run <pod-name> --image=<image-name> --port=<container-port>
$ kubectl run my-pod --image=nginx --port=80
```

2. View all the pods

(In default namespace)

```
$ kubectl get pods
```

(In All namespace)

```
$ kubectl get pods -A
```

For a specific namespace

```
$ kubectl get pods -n kube-system
```

For a specific type

```
$ kubectl get pods <pod-name>
```

```
$ kubectl get pods <pod-name> -o wide
```

```
$ kubectl get pods <pod-name> -o yaml
```

```
$ kubectl get pods <pod-name> -o json
```

3. Describe a pod (View Pod details)

```
$ kubectl describe pod <pod-name>
```

```
$ kubectl describe pod my-pod
```

4. View Logs of a pod

```
$ kubectl logs <pod-name>
```

```
$ kubectl logs my-pod
5. Execute any command inside Pod (Inside Pod OS)
$ kubectl exec <pod-name> -- <command>
```

DEPLOYMENT:

Deployment.yml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deploy
  labels:
    apptype: web-backend # Ensure this matches Service selector
spec:
  replicas: 4
  selector:
    matchLabels:
      apptype: web-backend # Ensure this matches Service selector
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        apptype: web-backend # Ensure this matches Service selector
    spec:
      containers:
        - name: my-app
          image: dhaarani15/simplewebapp:2
          ports:
            - containerPort: 7070
```

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  labels:
    apptype: web-backend # Make labels consistent
spec:
  type: NodePort
  ports:
    - targetPort: 8080
      port: 7070
      nodePort: 30002
```

selector:
apptype: web-backend

2. Create Deployment by executing above YAML file

```
$ kubectl create -f web-deploy.yml
# Do necessary modifications if exist, else create new
$ kubectl create -f web-deploy.yml
# Completely Modify Pod Template
$ kubectl replace -f web-deploy.yml
```

3. View Deployments

```
$ kubectl get deployments
$ kubectl get deploy
$ kubectl get deploy -o wide
$ kubectl get deploy <deployment-name> -o json
$ kubectl get deploy <deployment-name> -o yaml
```

4. View Deployment Description

```
$ kubectl describe deploy <deployment-name>
```

5. We can modify generated/updated YAML file

```
$ kubectl edit deploy <deployment-name>
## change replicas: count to any other value then (ESC):wq
```

We can modify our YAML file and then execute apply command

```
$ kubectl apply -f web-deploy.yml
```

We can Even scale using command also

```
$ kubectl scale deploy <deployment-name> --replicas=<desired-replica-count>
```

6. Delete Deployment

```
$ kubectl delete deploy <deployment-name>
$ kubectl delete -f web-deploy.yml
```

REPLICA SET:

```
kubectl create deployment webnginx2 --image=nginx:latest --replicas=1
```

```
kubectl scale deploy <deployment-name> --replicas=<desired-replica-count>
```

2. Create ReplicaSet by executing above YAML file

```
$ kubectl create -f rs-test.yml
# Do necessary modifications if exist, else create new
$ kubectl apply -f rs-test.yml
# Completely Modify Pod Template
$ kubectl replace -f rs-test.yml
```

3. View ReplicaSets

```
$ kubectl get replicaset
$ kubectl get rs
$ kubectl get rs -o wide
$ kubectl get rs <replica-set-name> -o json
$ kubectl get rs <replica-set-name> -o yaml
```

4. View ReplicaSet Description

```
$ kubectl describe rs <replica-set-name>
```

5. We can modify generated/updated YAML file

```
$ kubectl edit rs <replica-set-name>
```

```
## change replicas: count to any other value then (ESC):wq
```

```
# We can modify our YAML file and then execute apply command
```

```
$ kubectl apply -f rs-test.yml
```

```
## We can Even scale using command also
```

```
$ kubectl scale replicaset <replicaset-name> --replicas=<desired-replica-count>
```

6. Delete ReplicaSet

```
$ kubectl delete rs <replica-set-name>
```

```
$ kubectl delete -f rs-test.yml
```

Services (short name = svc):

Service is an abstraction that defines a logical set of pods and a policy to access them. Services enable network connectivity and load balancing to the pods that are part of the service, allowing other components within or outside the cluster to interact with the application.

Service Types: Kubernetes supports different types of services:

1. NodePort: Exposes the service on a static port on each selected node's IP. This type makes the service accessible from outside the cluster by the <NodeIP>:<NodePort> combination.
2. ClusterIP: Exposes the service on a cluster-internal IP. This type makes the service only reachable within the cluster.
3. LoadBalancer: Creates an external load balancer in cloud environments, which routes traffic to the service.

```
apiVersion: v1
```

```
kind: Service
```

```
metadata:
```

```
  name: my-service
```

```
  labels:
```

```
    app: my-service
```

```
spec:
```

```
  type: NodePort
```

```
  ports:
```

```
    - port: 9000
```

```
      targetPort: 8080
```

```
      nodePort: 30002
```

```
  selector:
```

apptype: web-backend

```
dhaaran@ITP-CC16-20:~$ minikube start
minikube v1.35.0 on Ubuntu 24.04 (amd64)
* Using the docker driver based on existing profile
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.46 ...
* Restarting existing docker container for "minikube" ...
* Preparing Kubernetes v1.32.0 on Docker 27.4.1 ...
* Verifying Kubernetes components...
  * Using image gcr.io/k8s-minikube/storage-provisioner:v5
  * Enabled addons: default-storageclass, storage-provisioner
  * Done! kubectll is now configured to use "minikube" cluster and "default" namespace by default
dhaaran@ITP-CC16-20:~$ kubectl get pod
NAME          READY   STATUS    RESTARTS   AGE
my-app        1/1     Running   1 (3m28s ago)    148m
my-pod        1/1     Running   1 (3m28s ago)    3h37m
my-rs-dpzm2   1/1     Running   1 (3m28s ago)    117m
my-rs-frp5    1/1     Running   1 (3m28s ago)    117m
my-rs-nc95h   1/1     Running   1 (3m28s ago)    117m
my-rs-zvmtq   1/1     Running   1 (3m28s ago)    117m
dhaaran@ITP-CC16-20:~$ kubectl get rs
NAME          DESIRED   CURRENT   READY   AGE
my-rs         4         4         4       122m
dhaaran@ITP-CC16-20:~$ sudo nano replicaset
[sudo] password for dhaaran:
dhaaran@ITP-CC16-20:~$ sudo nano replicaset.yml
dhaaran@ITP-CC16-20:~$ kubectl exec -it my-rs-dpzm2 -- /bin/bash
root@my-rs-dpzm2:/usr/local/tomcat# exit
exit
dhaaran@ITP-CC16-20:~$ ls
dhaaran@ITP-CC16-20:~$ docker-compose.yml pod.yml replicaset.yml
dhaaran@ITP-CC16-20:~$ sudo nano deployment.yml
dhaaran@ITP-CC16-20:~$ kubectl get node
NAME          STATUS    ROLES    AGE   VERSION
minikube      Ready     control-plane   25h   v1.32.0
dhaaran@ITP-CC16-20:~$ kubectl get pod
NAME          READY   STATUS    RESTARTS   AGE
my-app        1/1     Running   1 (21m ago)    166m
my-pod        1/1     Running   1 (21m ago)    3h55m
my-rs-dpzm2   1/1     Running   1 (21m ago)    135m
my-rs-frp5    1/1     Running   1 (21m ago)    135m
my-rs-nc95h   1/1     Running   1 (21m ago)    135m
my-rs-zvmtq   1/1     Running   1 (21m ago)    135m
dhaaran@ITP-CC16-20:~$ kubectl create deployment webgninx2 --image=nginx:latest --replicas=1
deployment.apps/webgninx2 created
dhaaran@ITP-CC16-20:~$ kubectl scale deploy webgninx2 --replicas=4
error: no objects passed to scale deployments.apps "webgninx2" not found
dhaaran@ITP-CC16-20:~$ kubectl scale deployment webgninx2 --replicas=4
error: no objects passed to scale deployments.apps "webgninx2" not found
dhaaran@ITP-CC16-20:~$ kubectl scale deployment webgninx2 --replicas=2
deployment.apps/webgninx2 scaled
dhaaran@ITP-CC16-20:~$
```

```
dhaaran@ITP-CC16-20:~$ nano deployment.yml
deployment.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment
  labels:
    name: deployment
spec:
  replicas: 4
  selector:
    matchLabels:
      apptype: web-backend
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        apptype: web-backend
    spec:
      containers:
        - name: my-app
          image: dhaaran115/simplewebapp:2
          ports:
            - containerPort: 7070
```

```
dhaaran@ITP-CC16-20: ~$ kubectl apply -f deployment.yml
NAME: my-service
NAMESPACE: default
TARGET PORT: 7070
URL: http://192.168.49.2:30002
Starting tunnel for service my-service.
NAME: my-service
NAMESPACE: default
TARGET PORT: 7070
URL: http://127.0.0.1:45359
Opening service default/my-service in default browser...
http://127.0.0.1:45359
Because you are using a Docker driver on linux, the terminal needs to be open to run it.
Stopping tunnel for service my-service.
dhaaran@ITP-CC16-20: ~$ curl http://192.168.49.2:30002
curl: (7) Failed to connect to 192.168.49.2 port 30002 after 0 ms: Couldn't connect to server
dhaaran@ITP-CC16-20: ~$ curl http://192.168.49.2:30000
curl: (7) Failed to connect to 192.168.49.2 port 30000 after 0 ms: Couldn't connect to server
dhaaran@ITP-CC16-20: ~$ kubectl delete deployment webginx2
deployment.apps "webginx2" deleted
dhaaran@ITP-CC16-20: ~$ kubectl apply -f deployment.yml
Warning: resource services/my-service is missing the kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl apply should only be used on resources c
reated declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
service/my-service configured
dhaaran@ITP-CC16-20: ~$ minikube service my-service
NAME: my-service
NAMESPACE: default
TARGET PORT: 7070
URL: http://192.168.49.2:30002
Starting tunnel for service my-service.
NAME: my-service
NAMESPACE: default
TARGET PORT: 7070
URL: http://127.0.0.1:34837
Opening service default/my-service in default browser...
http://127.0.0.1:34837
Because you are using a Docker driver on linux, the terminal needs to be open to run it.
Stopping tunnel for service my-service.
dhaaran@ITP-CC16-20: ~$ kubectl apply -f deployment.yml
service/my-service unchanged
dhaaran@ITP-CC16-20: ~$ curl http://192.168.49.2:30002
<!doctype html><html lang="en"><head><title>HTTP Status 404 - Not Found</title><style type="text/css">body {font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b {color:white;background-color:#525D76
} h1 {font-size:22px;} h2 {font-size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {color:black;} .line {height:1px;background-color:#525D76;border:none;}</style></head><body><h1>HTTP Status 4
04 - Not Found</h1><hr class="line" /><p><b>Type</b>: /b> Status Report</p><p><b>Description</b>: The origin server did not find a current representation for the target resource or is not willing to disc
lose that one exists.</p><hr class="line" /></body></html></dhaaran@ITP-CC16-20: ~$
```

Namespace (short name = ns):

namespace is a virtual cluster or logical partition within a cluster that provides a way to organize and isolate resources. It allows multiple teams or projects to share the same physical cluster while maintaining resource separation and access control.

To create a namespace:

\$ kubectl create namespace <namespace-name>

\$ kubectl create ns my-bank

To switch to a specific namespace: (make this as default type)

\$ kubectl config set-context --current --namespace=<namespace-name>

To list all namespaces:

\$ kubectl get namespaces

To get resources within a specific namespace:

\$ kubectl get <resource-type> -n <namespace-name>

\$ kubectl get deploy -n my-bank

\$ kubectl get deploy --namespace my-bank

\$ kubectl get all --namespace my-bank

To delete a namespace and all associated resources:

\$ kubectl delete namespace <namespace-name>

\$ kubectl delete ns my-bank

kubectl create ns mydeploy

kubectl apply -f deploy.yml -n mydeploy

.yml

apiVersion: v1

kind: Namespace

metadata:

name: my-demo-ns

```
chirinbanum@ChirinAnifa:~$ kubectl create ns mydeploy
namespace/mydeploy created
chirinbanum@ChirinAnifa:~$ kubectl apply -f deploy.yml -n mydeploy
error: the path "deploy.yml" does not exist
chirinbanum@ChirinAnifa:~$ kubectl apply -f deployment.yml -n mydeploy
error: the path "deployment.yml" does not exist
chirinbanum@ChirinAnifa:~$ kubectl apply -f deployment.yml -n mydeploy
deployment.apps/deployment created
chirinbanum@ChirinAnifa:~$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
deployment-7678898557-b2brr         1/1     Running   0           26m
deployment-7678898557-bwxpd         1/1     Running   0           135m
deployment-7678898557-cw957         1/1     Running   0           26m
deployment-7678898557-nzss5         1/1     Running   0           135m
deployment-7678898557-v96j6         1/1     Running   0           135m
my-app                               1/1     Running   0           5h19m
my-pod                               1/1     Running   0           6h38m
my-replicaset-66chn                 1/1     Running   0           4h49m
my-replicaset-f4f48                 1/1     Running   0           4h49m
my-replicaset-jdw7r                 1/1     Running   0           4h49m
my-replicaset-tq7gs                 1/1     Running   0           174m
chirinbanum@ChirinAnifa:~$ kubectl get deploy
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
deployment  5/5     5             5           135m
chirinbanum@ChirinAnifa:~$ kubectl get pod -n deploy
No resources found in deploy namespace.
chirinbanum@ChirinAnifa:~$ kubectl get pod -n mydeploy
NAME                                READY   STATUS    RESTARTS   AGE
deployment-7678898557-2f592         1/1     Running   0           114s
deployment-7678898557-bxf2r         1/1     Running   0           114s
deployment-7678898557-h6tg7         1/1     Running   0           114s
deployment-7678898557-lb4cb         1/1     Running   0           114s
deployment-7678898557-wpq8p         1/1     Running   0           114s
chirinbanum@ChirinAnifa:~$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
deployment-7678898557-b2brr         1/1     Running   0           27m
deployment-7678898557-bwxpd         1/1     Running   0           137m
deployment-7678898557-cw957         1/1     Running   0           27m
deployment-7678898557-nzss5         1/1     Running   0           137m
deployment-7678898557-v96j6         1/1     Running   0           137m
my-app                               1/1     Running   0           5h21m
my-pod                               1/1     Running   0           6h31m
my-replicaset-66chn                 1/1     Running   0           4h58m
my-replicaset-f4f48                 1/1     Running   0           4h58m
my-replicaset-jdw7r                 1/1     Running   0           4h58m
my-replicaset-tq7gs                 1/1     Running   0           176m
```

```
chirinbanum@ChirinAnifa:~$ kubectl get all --namespace mydeploy
NAME                                READY   STATUS    RESTARTS   AGE
pod/deployment-7678898557-2f592     1/1     Running   0           3m35s
pod/deployment-7678898557-bxf2r     1/1     Running   0           3m35s
pod/deployment-7678898557-h6tg7     1/1     Running   0           3m35s
pod/deployment-7678898557-lb4cb     1/1     Running   0           3m35s
pod/deployment-7678898557-wpq8p     1/1     Running   0           3m35s
chirinbanum@ChirinAnifa:~$ kubectl nano ns.yml
error: unknown command "nano" for "kubectl"
chirinbanum@ChirinAnifa:~$ kubectl nano ns.yml
error: unknown command "nano" for "kubectl"
chirinbanum@ChirinAnifa:~$ sudo nano ns.yml
[sudo] password for chirinbanum:
chirinbanum@ChirinAnifa:~$ kubectl apply -f ns.yml
namespace/my-demo-ns created
chirinbanum@ChirinAnifa:~$ sudo nano pod my-demo-ns
chirinbanum@ChirinAnifa:~$ sudo nano pod my-demo-ns
chirinbanum@ChirinAnifa:~$ kubectl apply -f ns.yml
```